A system for providing limited access to parts of texts of documents

This invention relates to servers, in particular web servers, which allow the client access to document which are also sold as paper documents, most typically books.

On many occasions, people would like to have a short look at a book, rather than read it all. For example, they may want to find some quote, read a little to see if they like the style, or lookup the name of a character in a book they already have read. It is specially useful if they can also search the books for some words. Other documents that people will find interesting include movie scripts, periodicals and even texts of daily newspapers. With today's computer power and World Wide Web (WWW) facilities, it would be reasonably easy to give such facilities online, by a server that has the texts of the books or the other documents and a search engine.

The problem is that if this is done, the publishers may loose revenue, because people will load documents off the server, rather than buy them ('publisher' here and in the rest of the document is used to mean 'a copyright owner'). Therefore, publishers would not allow their documents to be made available this way.

The solution comes from the observation that the server above is going to be useful even if it allows the clients to load only small parts from each document (here and in the rest of the text 'client' means 'network client', i.e. some other node on the internet, which is typically a person using a web browser). Therefore, it needs a mechanism that allows the clients to search all the text of the documents on one hand, but allows publishers to limit the access to

only small part of each document.

According the current invention (the *Constrained Document Server),* the 'engine' of the server is made of a database of the texts of documents (typically books) and of constraints for each document, and two software components: *searcher* and *extractor/filter*. When a client issues a request to the server (which would be some kind of query), the *searcher* searches the texts using standard techniques and identifies potentially matching documents. The identities of these documents are passed to the *extractor/filter*, which using the query extracts from each document pieces of text that match the query (the *extractor* functionality), and ensures that these pieces of text do not exceed the constraints of the document (the *Filter* functionality).

The two software components may be separate programs, or may be part of a single program. The *extractor* and the *filter* functionalities may be implement as two separate software components, but it seems that the *extractor* can be made more efficient if it is combined with the *filter*.

To complete the server, it will also need a front-end software which interacts with the clients. This includes interface to receive queries, maybe do some pre-processing on them, and then pass them to the *searcher*, and a component which receives the output of the *filter*, formats it in appropriate format and sends it back to the client. The front-end server may also identify the client, accept payments etc.

The identities of the documents can be passed between the components as one list, but they may also be passed in groups or each document on its own. As described below, a more

efficient implementation of the server actually passes identities of parts of the documents between the components.

To make the server useful, the constraints need to be easy to specify and to understand, so it will not require much expertise on the publishers side to constrain each document in the way they want it to be constrained.

An embodiment of the invention will now be described as an example.

For simplicity of the description, it is assumed that the syntax of the query that the user sends is simple, and after pre-processing the query consists of several *word-groups*, where each word-group consists of some words. For a document to match the query, it has to match all the word-groups. A word-group matches a continuous text that contains all the words in the group. The number of words in each group is variable, as is the number of groups.

The example server is intended to give reliable responses only for queries that result in a response which is much below the constraints, so when the response is close to exceed the constraints it can be unreliable.

The constraints in the example server are made by logical combination of primitive constraints. A primitive constraint is of the form <CHUNK> [PER] [IN <BLOCK>](square brackets denote optional part of the expression. Angle brackets denote a variable expression.) <BLOCK> and default to all the document. The

meaning of the constraint is that inside the <BLOCK>, the client can get at most <CHUNK> per each .

The values of the CHUNK, SEGMENT and BLOCK are specified by a phrase of the form:

[<LOCATION>] [<Number>] <ELEMENT> [<Range>]

Where:

LOCATION is one of

FIRST, SECOND, THIRD, PENULTIMATE, LAST, MIDDLE

Number is a positive real number

ELEMENT is one of

PAGE, PARAGRAPH, LINE, CHAPTER, PERCENT, CHARACTER, HALF, THIRD, QUARTER, ALL

or the plural form of any of these except ALL

Range is either a number or a pair of numbers separated by an hyphen.

For example:1 PAGE PER 1 CHAPTER - the client can get one page per each chapter. Note that because the IN <BLOCK> clause is omitted, this applies to all the document. 2 PARAGRAPHS PER PAGE - The client can get at most 2 paragraphs per page.3 PAGES - the client can get at most 3 pages from all the document.

The IN <BLOCK> clause allows the publishers to restrict access to part of the document, e.g. PAGE 1-2 PER CHAPTER IN FIRST 3 CHAPTERS - allows the client to see the first 2 pages of the first three chapters.3 PARAGRAPHS PER 2 PAGES IN MIDDLE 2 QUARTERS - allows the client to see at most 3 paragraphs per each two pages in the

second and third quarters.The primitive constraints can be combined into a *constraint expression* byAND or OR. AND means both the preceding and following primitive constraints must be satisfied, OR means that at least one of them must be satisfied. Brackets can be used to ensure the right conjunction. Examples:

10.5 LINES PER PAGE AND 5 PERCENT - The client can get at most 10.5 lines per page but the total of text must be less than 5% of all the document.

3 PAGES PER CHAPTER IN FIRST HALF OR 1 PAGE PER CHAPTER IN CHAPTERS 7-11Allows the client to get up to 3 pages per chapter from the first half of the document and 1 page per chapter from chapters 7-11.

3 PAGES OR (1 PAGE PER CHAPTER AND 2 PAGES IN SECOND HALF)

Allows the client to receive anything up to 3 pages, or more than 3 pages provided it contains less than 1 page per chapter and less than 2 pages from the second half of the document.

This syntax is flexible enough to allow the publishers to tailor the constraints to their requirements, but is still very easy to implement (including error checking), and the meaning of the expressions is obvious to humans.

The terms PAGE, LINE, PARAGRAPH etc. are somewhat ambiguous in documents that are in digital form, but it would reasonably easy to define them. In fact, any arbitrary mapping that is not completely counterintuitive (most simply, numeric translation: LINE = 80 characters, PARAGRAPH = 600 characters, page = 5000 characters) would be good enough, because the publisher can simply put the constraint, do some requests and if they believe they get too much or too little, adjust the numbers in the constraints as appropriate. The borders of CHAPTERS will have to be explicitly specified by the

publishers. A document can have several constraint expressions for different levels of service, one (or none) for free access and others for restricted access, e.g. for subscribers or for paid access. For example, a document may have these constraint expressions:

Free: 1 PAGE PER CHAPTER AND 3 PAGES

subscriber: 2 PAGES PER CHAPTER AND 10 PAGES

one-time payment 1$: CHAPTER IN FIRST 10 CHAPTERS

one-time payment 5$: ALL IN FIRST HALF

Which means: all clients can get 1 page per chapter, up to 3 pages from all the document. Subscribers can get 2 pages per chapter and up to 10 pages in all. Clients can get any of the first 10 chapters for 1$ each, and all the first half for 5$.

The input interface of the example server can vary. One possible interface is presenting the user with several input areas, and the words in each input area comprise a word-group. Another possibility is entering each group in brackets, i.e.

(X1 X2) (X3 X4) where Xn is some word.

The input interface pre-processes the input from the user into a format recognisable by the *searcher* and the *extractor/filter*, adds an identifier which tells the *extractor/filter* the access level, and hence which constraint expressions to use (The *searcher* can normally ignore the access level).

The *searcher* is a simple index-based searcher, which relies on an index of words pointing to documents for fast search. When a document is added to the database, the server goes through it and for each word adds the document to the entry of the word in the index. When it receives a query, it retrieve from the index the entry for each word in each word-group in

the query, and then checks which documents appear in all these lists, and pass these documents to the *extractor/filter*. In principle, the *searcher* can ignore the grouping of words.

The *extractor/filter* tries to match the word-groups. For each word-group it takes the most infrequent word and search for it, starting from the beginning of the document in the first time it searches for a word-group, or from the end of the previous match. If it finds this word, it checks what is the maximum amount of text that the client is allow to receive from this segment of the document. It uses this value as a limiting distance, and searches for the other words in the word-group within this distance from the match of the most infrequent word. If it finds all the words in the word-group within this limit, it marks the smallest region of the text that contains all the words plus some small margin as a match. Otherwise, it finds the next occurrence of the most infrequent word and repeat the same process.

When it finds a match for a word-group, it check if the accumulated matches up to this point exceed any of the constraints. If it does, it checks whether the current match has a large contribution to the exception. If it does, it ignores the current match, and tries to find another match. If the contribution of the current match is small, it skips to the end of the segment for which the exception happens, and continues to search. For example if the constraint that has been exceeded is two pages per chapter, if the current match is itself 1.5 pages, it will continue to search in the same chapter for a smaller match, while if the current match is only 0.25 page, it will skip to the next chapter. The exact threshold between skipping to the next segment or not is not important, because of the assumption that when the response is close to the constraint, the result does not have to be reliable.

Once a match for a word-group is found, the *extractor/filter* tries to match the next word-group, or, if it is the last word-group, tries to match again the first group. It repeats the cycle until it reaches the end of the document with all the word-groups, or extracted more text than the client is allowed to receive from all the document.

Once the *extractor/filter* reached the end of the document, it passes the extracted pieces of texts to the output interface, which formats them in an appropriate format and sends them to the client. If any of the constraints has been exceeded, the *extractor/filter* may also pass a message to the client that it happened (depending on the server configuration). This message also includes the constraint expression, with the constraint that has been exceeded highlighted, so the client, assuming it is actually a person, can have an idea why their query matched too much of the text. If the document has constraint expressions for a one-time payment which are higher than the payment of the current level, the *extractor/filter* may also pass a list of these expressions and the associated payments.

The implementation which is described above is simple, but has the problem that in many cases, the number of documents which the *searcher* will find will be quite large, because whole documents contain large range of words. This will mean that the amount of text that the *extractor/filter* will have to search would be large. Since the *extractor/filter* searches all the text, this may take a long time. A way to improve the performance of the server is to keep each documents as collection of smaller pieces of texts (*sub-documents*). Instead of indexing whole documents, the sub-documents are indexed individually. When the *searcher* receives the query, it finds for each word-group in the query all the sub-documents that match it, and passes to the *extractor/filter* the sub-documents of those documents that match each word group in at least one of their sub-documents. This approach both prevents

the *searcher* from passing on documents where a word-group matches over large portion of the text, because none of the sub-documents will match it, and for each document that is passed, the *extractor/filter* will have to search only those sub-documents that the *searcher* identified.

The size of the sub-documents can be manipulated to maximise the performance of the server, and can be different for each document. A lowest minimum is set by the largest chunk that the user is allowed to receive, and to prevent missing matches across sub-documents boundaries, they will need to have some overlap, except when the boundaries correspond to BLOCK boundaries in the constraints. For example, if the largest chunk that the user is allowed to get is page, the document may be kept in sub-documents where each sub-document contains two pages of text, and has one page overlap with the next sub-document, i.e. the first sub-document contains pages 1 and 2, the second contains pages 2 and 3, the third pages 3 and 4, etc. This doubles the amount of storage for the document, but saves the *extractor/filter* a lot of work. Less disk-usage with reduced performance can be achieved by larger sub-documents with the same overlap size, e.g. each sub-document containing a chapter plus the first page of the next chapter.

In cases where the chunks that the user is allowed to get are large (e.g. half a book) it is more difficult to get this improvement, but it is possible to add to the definition of the query the restriction that a word-group must match on some maximum chunk of text (e.g. a chapter), and then the size of sub-documents need never be bigger than twice this chunk size to prevent missing matches across boundaries, assuming the sub-documents overlap as described above. The additional restriction will have very little effect on the usefulness of the server.

To avoid spending too much time on queries that match too many documents (or sub-documents), the server need to have a limit of the amount of text that the *extractor/filter* will search, which will vary according to the level of service. When the amount of text that *searcher* identifies as potentially containing matches is above the limit, the client will be informed that their query is too expensive, and optionally asked if they are ready to pay for it.

A useful extension to the server will be to allow the users to specify some subset of the documents to search in, typically by specifying some category. For books, that would include all the categories that are typically used by book sellers and libraries, like books by specific author(s), books that contain some specific words in their title, books of specific genre, books from specific year(s) of publication, country of publication etc. In addition, it may include any category that seem to be useful, e.g. all books that were nominated to the booker prize, books that were made into movies etc. For movie scripts, it will include movies with specific people in their cast.

The server can be made more useful by additional several relatively simple extensions to the syntax of queries:

1) Allow the client to specify in which segment of the document to find the word-group (where segment has the same semantics as in the constraints). If the documents are divided to sub-documents, the *searcher* will pass only sub-documents in this part of the document, otherwise it will ignore it. The *extractor/filter* will use it to limit the search for the word-group. For example, the client may enter:

(HORSE DINOSAUR) INSIDE FIRST 2 CHAPTERS

Which is pre-processed by the input interface to mean a word-group containing HORSE and DINOSAUR, to be found in the first 2 chapters. The *searcher* finds all the documents (or sub-documents) that contain HORSE and DINOSAUR. The *extractor/filter* searches each document for DINOSAUR (because it is the less frequent word) in the first 2 chapters, and if it finds it searches for the word HORSE around this match.

2) Allow the client to specify the length of continuous text in which the word-group must be matched. For example, the client may enter:

(HORSE DINOSAUR) INSIDE FIRST 2 CHAPTERS WITHIN 2 LINES

Which is processed the same as the previous example in (1), except that the search for HORSE is limited to 2 lines from the match of DINOSAUR. It is assumed that this limitation is smaller than the limit set by the constraints. If it larger than the constraints, it is ignored.

3) Allow each word to be a phrase, i.e. a sequence of character that must match exactly, even if it contains spaces. On the client side, that would typically be done enclosing the phrase in double quotes, e.g. "bright yellow balloon". Because the *searcher* uses a word-index, it treats a phrase as several words (with the example these would be 'bright', 'yellow' and 'balloon'), and retrieves documents (or sub-documents) which contain all the words in the phrase but not necessarily the phrase itself. The *extractor/filter* treats all the

phrase as a single word, and hence only matches of the exact phrase are returned to the client.

4) Allow OR relations. Allow a word to be replaced by an expression meaning "any of these words", and allow a word-group to be replaced by an expression meaning 'any of these word-groups'. For example, square brackets can delimit an OR group, e.g. [orange apple banana] may mean 'orange', 'apple' or 'banana', and [(horse black) (donkey white)] may mean either 'horse' and 'black' or 'donkey' and 'white'. This is easy to implement in both the *searcher* and the *extractor/filter.*

5) Allow wild-cards, e.g. '*' as matching everything, so something like STONE* means any word starting with stone. This is done by finding in the index all the words that match the pattern, and then using them as an OR group.

The server may be also made to work with the 'canonical forms' of words. For example, plural words may be converted to the single form, and verbs may be converted to their root. This is trivial change to the *searcher* and makes it more efficient, but means it find more documents. For the *extractor/filter* it adds complexity and make it slower, because for each word it has search for the canonical form and all the derivatives. For regular verbs and nouns it can simply search for the canonical form and check if it exact match or followed by an acceptable suffix, but for irregular verbs and nouns it will have to search for the irregular form as well. If the server is used with a language where the plural or the tense is not a suffix, it will make it even more complex. If the server is made to use a canonical form, it can also have the option to switch it off.

From the point of view of the publisher, a possible improvement to the server is a database indexed on publisher and keywords (arbitrary sequence of letters), which specifies some value for each keyword and publisher, and an extension to the syntax of the constraints allowing a reference to keyword, e.g. by starting a word with a '$'. For example, assuming the database contains this entries

Publisher1 Suspense => 4 PAGES IN FIRST 3 QUARTERS

Publisher1 A-lot => 10 pages

Publisher1 A-little => 1 page

Then when Publisher1 adds a document to the database, they can specify the constraints as $Suspense. When the *extractor/filter* sees the '$' sign, it will take the publisher of the document and the rest of the word, i.e. Publisher1 and Suspense, and searches the database to find 4 PAGES IN FIRST 3 QUARTERS. Publisher1 can also specify as a constraint

$A-lot IN FIRST HALF OR $A-little IN SECOND HALF

Which is expanded as above to 10 PAGES IN FIRST HALF OR 1 PAGES IN SECOND HALF.

This extension has two advantages:

1) It allows the publishers to simplify constraints specification, because they can put types

of documents as keywords in the database, and then just specify the document type (preceded by '$') as the constraints. This is demonstrated by the $Suspense example above.

2) It make it easier to control the constraints. For example, Publisher1 may change the entry in the database to

Publisher1 A-lot => 8 PAGES

and all the constraints that has $A-lot in them automatically change.